HTTP MULTIPLEXOR/DEMULTIPLEXOR

Cross-Reference to Related Applications

This application claims priority from U.S. provisional patent application, Serial No. 60/239,552, entitled "HTTP Multiplexor/Demultiplexor," filed on October 10, 2000, the disclosure of which is herein incorporated by reference.

Technical Field

The present invention relates generally to data transmission on computer networks, and more particularly to a Hypertext Transfer Protocol (HTTP) Multiplexor/Demultiplexor.

Background of the Invention

The Internet has experienced explosive growth in recent years. The emergence of the World Wide Web has enabled millions of users around the world to download easily web resources containing text, graphics, video, and sound data while at home, work, or from remote locations via wireless devices. These web resources often are large in size and therefore require a long time to download, causing the user delay and frustration. Delay often causes users to abandon the requested web page and move on to another web page, resulting in lost revenue and exposure for many commercial web sites.

One cause of delay is accumulation of Hypertext Transfer Protocol (HTTP) requests within a Transfer Control Protocol (TCP) buffer of a server socket at a server. When a user requests a web page, a web browser sends HTTP requests to a server socket via an established TCP connection. When the server does not process requests to a

socket quickly enough, HTTP requests build up in the TCP buffer for that socket, resulting in processing delay in that socket.

It would be desirable to provide a system, method, and device for reducing buffer congestion and delay in server response time.

Summary of the Invention

A system, method and device for multiplexing and demultiplexing HTTP requests are provided. The method may include receiving HTTP requests from a client and routing those requests to a plurality of sockets on a server. The requests may be routed based on socket response time, the type or size of data being requested, or on other parameters related to the HTTP requests. The method may also include receiving HTTP responses over a plurality of connections from the server and routing those responses to the client.

According to another aspect of the invention, the method typically includes at an intermediate networking device, listening for a series of HTTP requests from an originating client, receiving the series of HTTP requests from the originating client, demultiplexing the series of HTTP requests into discrete HTTP requests, and sending each discrete HTTP request to an optimal server socket. The method further includes listening for HTTP responses from a plurality of server sockets, receiving the HTTP responses from the plurality of server sockets, multiplexing the HTTP responses from the plurality of server sockets into a series of HTTP responses, and sending the series of HTTP responses to the originating client.

The system typically includes a server, a client configured to connect to the server via a computer network, and a computer networking device positioned intermediate the server and the client on the computer network. The computer networking device typically has an HTTP multiplexor/demultiplexor configured to receive HTTP requests from the client and to distribute those requests over a plurality of TCP connections to a plurality of corresponding sockets on the server.

The device typically includes an HTTP multiplexor/demultiplexor configured to receive HTTP requests from a client and to distribute those requests to a plurality of sockets on a server. The device typically is further configured to receive HTTP responses from the plurality of sockets on the server and to route those responses to the client.

Brief Description of the Drawings

Fig. 1 is a schematic view of a prior art network configuration.

Fig. 2 is a schematic view of an HTTP multiplexor/demultiplexor system according to one embodiment of the present invention.

Fig. 3 is a schematic view of the HTTP multiplexor/demultiplexor system of Fig. 2, showing HTTP requests and responses.

Fig. 4 is a schematic view of a client computing device of the system of Fig. 2.

Fig. 5 is a schematic view of one embodiment of a networking device of the system of Fig. 2.

Fig. 6 is a schematic view of another embodiment of a networking device of the system of Fig. 2.

Fig. 7 is a flowchart of a method of demultiplexing and multiplexing HTTP requests and responses according to one embodiment of the present invention.

Detailed Description of the Invention

Referring initially to Fig. 1, a prior art networking system is shown generally at 5. Prior art system 5 includes a plurality of remote clients 5a and a server 5b. In prior art system 5, a single Transport Control Protocol (TCP) connection is established between each remote client 5a and server 5b. Each TCP connection is established between a single socket on each remote client and a corresponding socket on the server, such that a one-to-one socket ratio is established. Remote clients 5a send Hypertext Transfer Protocol (HTTP) requests via established TCP connections to a TCP buffer associated with a server socket. Requests received at the TCP buffer are processed only as quickly as the server can respond to them. Often, requests build up in the buffer because the server cannot respond to them quickly enough, and server-side delay (also referred to as latency) often results. This is inefficient and frustrating, and may cause a user to abandon downloading the page.

In Fig. 2, a system for processing HTTP requests according to one embodiment of the present invention is shown generally at 10. System 10 typically includes a plurality of remote clients 12 configured to communicate with servers 14 via computer network 16. To access web resources located at Uniform Resource Indicators (URIs) on web site 18, remote client 12 sends HTTP requests via a networking device 20

to server 14 and waits for HTTP responses to return. Networking device 20 includes an HTTP multiplexor/demultiplexor 22 configured to route HTTP requests from a single socket on remote client 12, to a plurality of sockets on server 14. Typically, HTTP requests are routed to an optimal server socket with optimal response time, in order to ensure efficient processing of the requests.

Referring now to Fig. 4, remote client 12 typically is a personal computer including a processor 13a coupled to a communications bus 13b. A mass storage device 13c, such as a hard drive, CD ROM drive, tape drive, etc., and a memory 13d are also linked to the communications bus 13b. Memory 13d typically includes random access memory (RAM) 13e, and read-only memory (ROM) 13f. ROM 13f typically includes a basic input/output system (BIOS) 13g, which is configured to start up and operate basic functions of the remote client. Remote client 12 typically is configured to access computer network 16 via a network interface 13h. Alternatively, remote client 12 may be a portable data assistant, web-enabled wireless device, mainframe computer, or other suitable computing device.

Remote client 12 typically is configured to run an operating system (OS) 13i to manage programs or applications. Operating system 13i is stored in mass storage device 13c. Examples of suitable operating systems include UNIX, Windows, MacOS, VMS, and OS/2, although virtually any suitable operating system may be used. Remote client 12 includes a browser program 13j stored in mass storage device 13c configured to display requested web resources to a user of remote client 12. Exemplary browser programs 34 include the Netscape browser commercially available from Netscape

Communications Corporation of Santa Clara, California and the Internet Explorer browser commercially available from Microsoft Corporation of Redmond, Washington.

Server 14 also typically is a computer similar to that shown in Fig. 4. Server 14 typically includes a server program configured to communicate with remote clients using the HTTP protocol. The server program typically is configured to receive HTTP requests, and, in response send HTTP responses to browser 13j on remote client 12 via computer network 16.

Computer network 16 typically is a wide area network (WAN) such as the Internet and computer network 24 is a local area network (LAN). Typically, network 16 operates using TCP/IP protocols, although other protocols suitable for carrying HTTP requests and responses may be used.

Web site 18 typically includes a collection of web resources or URIs typically located at a web address called a URL (Uniform Resource Locator). The term "web resource" refers generally to a data resource that may be downloaded by a web browser. Web resources may include web pages, code, graphics, video, sounds, text, and/or other data. Web resources may be static (e.g. stored file) or dynamic (e.g. dynamically generated output). Web resources may be stored on and served by a single server 14 or a number of servers 14, as shown in Figs. 1 and 2. For example, images may be stored on one server while code may be stored in another server, alternatively, copies of images and code may be stored on multiple redundant servers.

As shown in Fig. 5, networking device 20 typically includes a controller 20a having a memory 20b and processor 20c linked by a bus 20d. Also coupled to bus

20d is a mass storage device 20e including a multiplexor/demultiplexor 22, which may also be referred to as a "mux/demux." Networking device 20 also typically includes a network interface 20f coupled to bus 20d and to an external network connection to computer network 16. Network interface 20f is configured to enable networking device 20 to communicate with remote client 12 via WAN computer network 16 and with server 14 via LAN computer network 24. An example of a suitable network interface is the Intel Ethernet Pro 100 network card, commercially available from Intel Corporation of Santa Clara, California.

In Fig. 6, another embodiment of a networking device according to the present invention is shown generally at 20′. Networking device 20′ typically includes an integrated circuit board 20g. The integrated circuit board contains a bus 20h connecting a network interface 20i, memory 20j, processor 20k, Application Specific Integrated Circuit (ASIC) 20m, and mass storage device 20n. Network interface 20i is configured to enable networking device 20′ to communicate with remote client 12 via computer network 16 and with server 14 via LAN 24. ASIC 20m typically contains a multiplexor/demultiplexor 22. ASIC 20m, processor 20k, and memory 20j form a controller 20p configured to process requests for web resources according to the methods described below. It will be appreciated that the embodiments of networking device 20, 20′ may be a stand-alone network appliance or may be integrated with a web server.

Networking device 20 typically is connected to server 14 via LAN 24. Because device 20 is connected to server 14 via LAN 24 and remote client 12 via WAN 16, networking device 20 may be considered a "server-side" proxy server. A proxy

server is a program or device that acts as an intermediary between a browser and a server. Networking device 20 acts as an intermediary by receiving HTTP requests from remote clients 12 and sending those requests to a socket on server 14, and by receiving server-generated HTTP responses and sending those responses to the remote client that originated the requests.

Networking device 20 includes a software or firmware multiplexor/demultiplexor 22 configured to route requests from a single remote client to a plurality of sockets on server 14, and to route server responses from various sockets back to the originating remote client. As shown in Fig. 3, each remote client 12 has an associated network connection 26, 28, 30 established with multiplexor/demultiplexor 22 via WAN 16. Remote clients 12 are typically configured to send HTTP requests and receive HTTP responses via connections 26, 28, and 30. Each server 14 has an associated network connection 32, 34, 36 established with multiplexor/demultiplexor 22 via LAN 24. Servers 14 are typically configured to receive HTTP requests and send HTTP responses via connections 32, 34, and 36. It will be appreciated that multiplexor/demultiplexor 22 is configured to establish additional connections with additional remote clients and servers.

Typically connections 26, 28, 30, 32, 34, and 36 are persistent TCP connections. Persistent TCP connections are connections that remain open until explicitly commanded to close or until the server times-out the connection. Alternatively, a connection other than a persistent TCP connection may be used.

HTTP multiplexor/demultiplexor 22 is configured to receive a series of HTTP requests A over a single connection 26 from a remote client 12 at a corresponding remote client-side socket 26a. As used herein, the term socket refers to a port, buffer, logical node, or object configured to receive data in the HTTP format from a remote device via a network connection, and is not limited to a "socket" as defined in the Unix operating system environment.

HTTP multiplexor/demultiplexor 22 is also configured to demultiplex the series of requests A into discrete requests A_1-A_3 and to transport each discrete request A_1-A_3 to one of a plurality of server-side sockets 32a, 34a, and 36a. This process is referred to as demultiplexing because a series of requests from a single TCP connection is broken up and routed over a plurality of TCP connections to a plurality of server sockets. Typically server-side sockets 32a, 34a, 36a of mux/demux 22 are connected to corresponding server sockets 32b, 34b, 36b of server 14 via respective connections 32, 34, 36, via LAN 24.

HTTP multiplexor/demultiplexor 22 of one embodiment of the present invention is configured to route or distribute the incoming requests A from a single remote client-side socket such as 26a to a plurality of server-side sockets 32a, 34a, and 36a. In doing so, multiplexor/demultiplexor 22 is configured to route each of HTTP requests A to an optimal server socket, which typically is a least busy server socket. To determine the optimal server socket, multiplexor/demultiplexor 22 may be configured to detect the response time at each server socket 32b, 34b, 36b by monitoring server-side sockets 32a, 34a, and 36a. The server socket with the fastest response time may be

determined to be the least busy server socket. Alternatively, another method may be used to determine the optimal server socket. By adjusting the flow of requests away from slow, congested server sockets and toward fast congestion-free server sockets, the multiplexor/demultiplexor is able to increase the overall efficiency and response time of server 14.

In addition, the HTTP multiplexor/demultiplexor may be configured to determine the type of HTTP request being made and/or the type of data being requested and accordingly route the request to an optimal server-side socket, based on the requested data type or HTTP request type. For example, all image requests may be handled by a predetermined set of sockets on server 14a, while all HTML requests may be handled by a predetermined set of sockets on server 14b. In addition, all HTTP 1.0 requests may be detected by multiplexor/demultiplexor 22 and routed to an optimal socket on server 14a, while all HTTP 1.1 requests may be detected and routed to an optimal socket on server 14b.

Server 14 is configured to respond to the many incoming HTTP requests by sending out appropriate HTTP responses B, which may contain data requested by the HTTP requests. HTTP multiplexor/demultiplexor 22 is configured to receive these responses at server-side sockets 32a, 34a, and 36a and route these responses back to the appropriate remote client-side socket, such as 26a, from which the corresponding HTTP request originated. For example, in the example shown in Fig. 3, HTTP requests A_1, A_2, and A_3 all originate from remote client 12a, and are routed to server sockets 32b, 34b, and 36b, respectively, by HTTP multiplexor/demultiplexor 22. In response, HTTP

responses B_1, B_2, and B_3 are sent from server sockets 32b, 34b, and 36b over persistent TCP connections 32, 34, and 36 to server-side sockets 32a, 34a, and 36a, at which point the responses are all routed back to remote client-side socket 26a for delivery to remote client 12a via TCP connection 26. This process is referred to as multiplexing because discrete responses from a plurality of sockets are combined into a series of responses and sent over a single TCP connection to remote client 12a.

Thus, HTTP multiplexor/demultiplexor 22 is configured to take a single series of HTTP requests received via a single socket and route the requests to a plurality of different sockets on server 14, and route responses from the various server sockets back to the remote client, such that the performance of server 14 may be optimized.

Although multiplexing and demultiplexing has been described in detail with reference only to remote client 12a, it will also be appreciated that HTTP multiplexor/demultiplexor 22 is configured to simultaneously demultiplex and multiplex HTTP requests and responses going to and from a plurality of remote clients 12 via a plurality of remote client-side sockets 26a, 28a, 30a.

Turning to Fig. 7, a method 100 may be practiced according to the present invention. The steps of method 100 are typically accomplished by networking device 20, utilizing remote client 12, server 14, WAN 16, and LAN 24. Alternatively, the method may be accomplished by dedicated software on server 14, or by some other suitable software or hardware device. At 102, the method typically includes establishing persistent TCP connections between multiplexor/demultiplexor 22 and a plurality of sockets on server 14 typically via LAN 24. At 104, the method typically includes

establishing persistent TCP connections between multiplexor/demultiplexor 22 and one or more remote clients 12 via WAN 16. At 106, the method further includes listening for a series of HTTP requests from an originating remote client 12 and/or for HTTP responses from various server sockets.

When a series of HTTP requests is detected at multiplexor/demultiplexor 22, method 100 continues, at 108, to receive the series of HTTP requests from originating remote client 12 at a single remote client-side socket. Method 100 further includes, at 110, demultiplexing the series of HTTP requests into a plurality of discrete HTTP requests. At 112, the method further includes routing the series of requests to a plurality of sockets on an associated server, typically by sending each discrete HTTP request to an optimal server socket.

Prior to step 112, method 100 may also include monitoring server sockets 32b, 34b, and 36b and determining an optimal server socket. The multiplexor/demultiplexor typically determines an optimal server socket by determining a server socket with a least-lengthy response time. Alternatively, the optimal server socket may be determined by determining a last-accessed server socket, determining a server socket with the fewest number of unfulfilled requests, determining the type or size of data being requested or other parameters related to the HTTP requests, or by weighing all or some of these conditions. By determining an optimal server socket, the multiplexor/demultiplexor is able to send a discrete HTTP request to the optimal server socket.

When HTTP responses are detected at the multiplexor/demultiplexor at step 106, method 100 proceeds to 114, and includes receiving HTTP responses from various server sockets. The multiplexor/demultiplexor typically is able to determine the destination of the HTTP responses. At 116, the method includes multiplexing the HTTP responses generated in response to requests from an originating remote client into a series of HTTP responses bound for the originating remote client. At 118, method 100 includes sending the series of responses to the originating remote client.

When there is a new remote client or server detected at 120, the method includes returning to step 104 to establish a persistent TCP connection with the new remote client, or returning to step 102 to establish a persistent TCP connection with the new server, respectively. It will also be appreciated that HTTP multiplexor/demultiplexor 22 may be configured to establish a plurality of TCP connections with a plurality of servers and a plurality of remote clients, and therefore may be configured to handle HTTP requests and HTTP responses from multiple servers and remote clients at once.

According to another embodiment of the method, the method may include receiving a series of HTTP requests originating from a single remote client and routing those requests to a plurality of sockets on a server. The method may also include receiving HTTP responses over a plurality of TCP connections from the server and routing those responses to a single remote client. As used above, the term "single remote client" refers to the fact that a plurality of requests originate from a single client. Of course, it will be appreciated that the method may be practiced with more than one remote client.

Use of the above described HTTP processing system 10, multiplexor/demultiplexor 22, and methods, in effect, reduce latency of server 14 in responding to HTTP requests because persistent connections between the server sockets and server-side sockets of multiplexor/demultiplexor 22 enable improved distribution of requests to optimal server sockets to maximize the efficiency of server 14.

While the present invention has been particularly shown and described with reference to the foregoing preferred embodiments, those skilled in the art will understand that many variations may be made therein without departing from the spirit and scope of the invention as defined in the following claims. The description of the invention should be understood to include all novel and non-obvious combinations of elements described herein, and claims may be presented in this or a later application to any novel and non-obvious combination of these elements. Where the claims recite "a" or "a first" element or the equivalent thereof, such claims should be understood to include incorporation of one or more such elements, neither requiring nor excluding two or more such elements.